计算机学院专业课程

计算机组成

流水线处理器
参考设计方案与代码架构

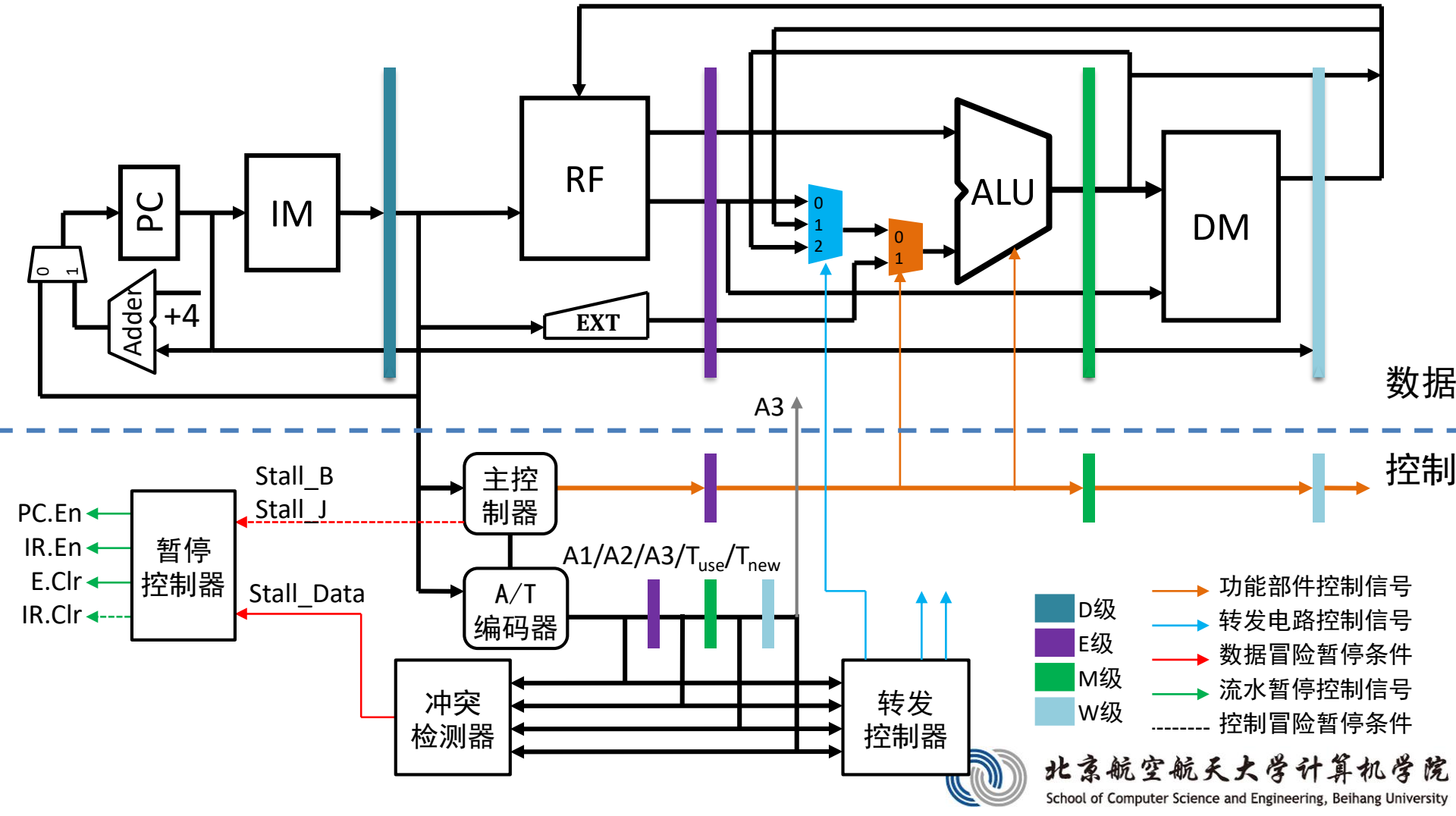
高小鹏

北京航空航天大学计算机学院

系统总体架构

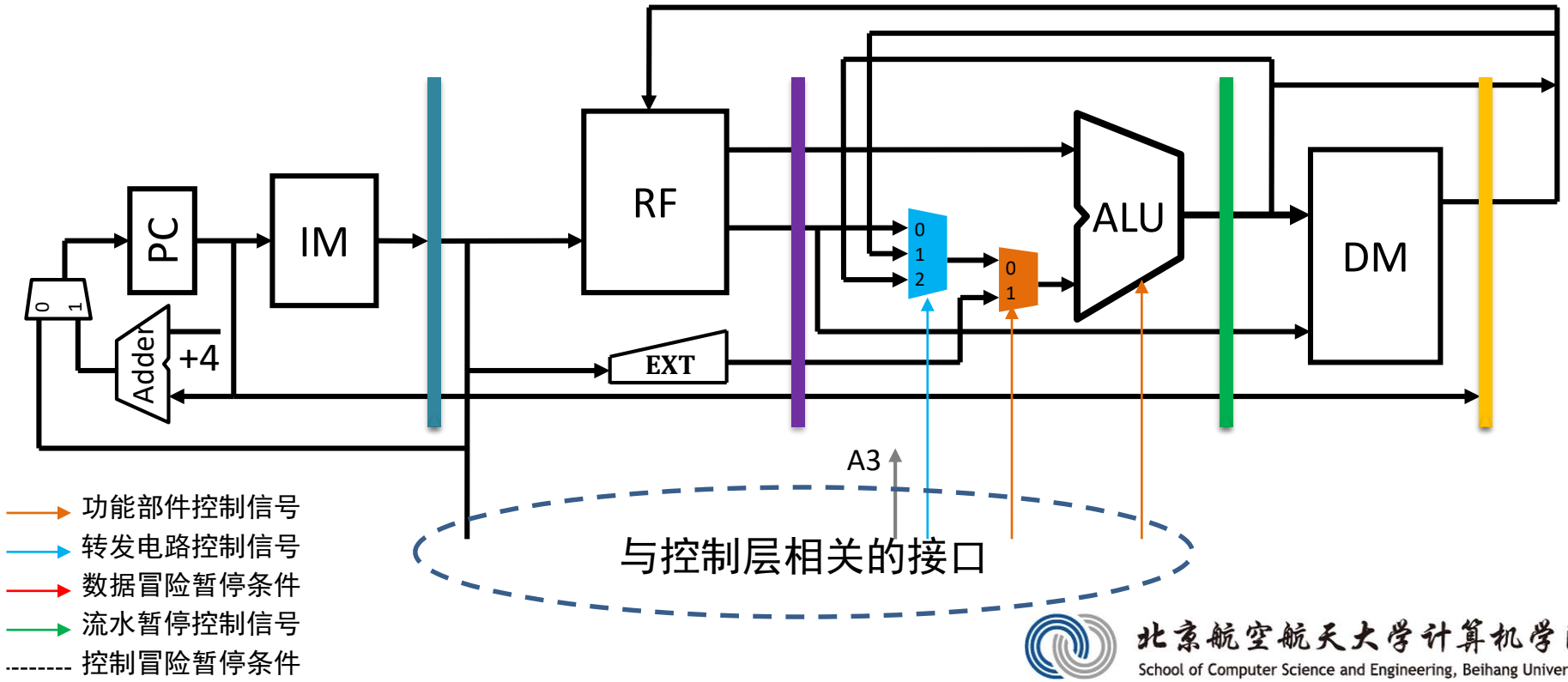
TIPS
仅示意了E级
部分控制信号

- 从总体上，系统分为2个层面：数据层，控制层
- 两层之间：指令、各种控制信号、A3（后面会解释）



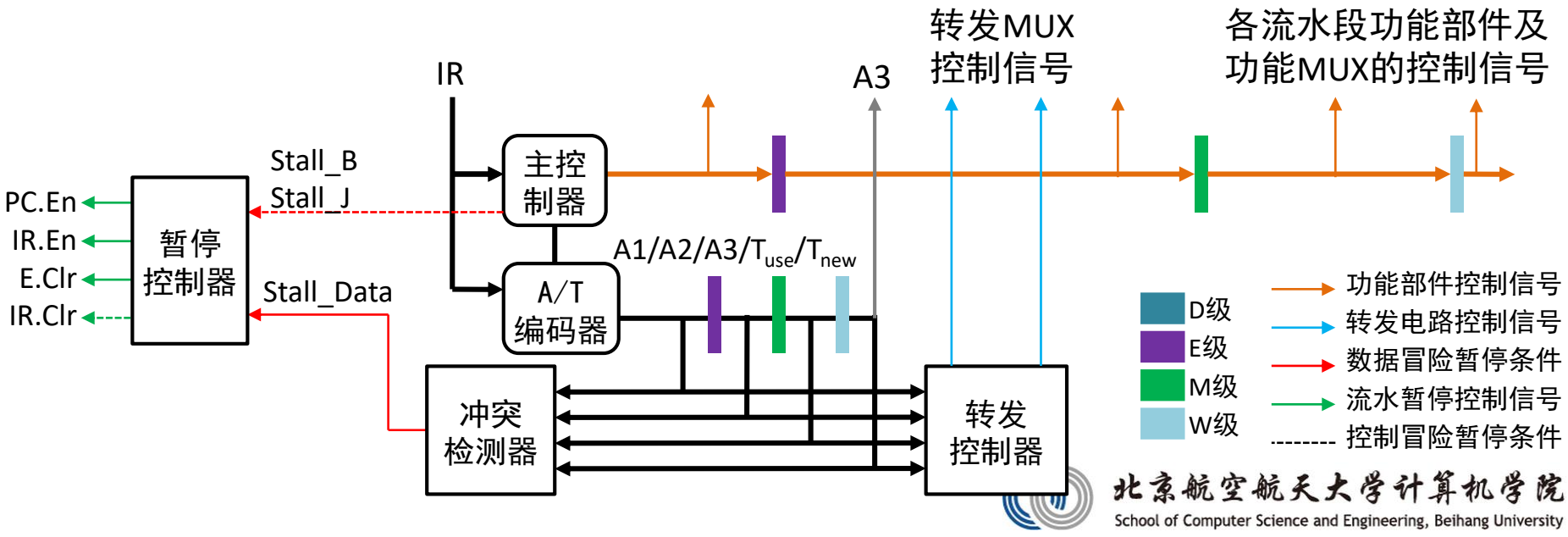
数据层

- 数据层：面向指令的数据流
  - 包含功能部件、功能部件间的数据连接关系、数据信息的流水寄存器
  - 数据层是“机制与策略分离”的机制部分
    - 机制：指令执行所需要的各类功能部件



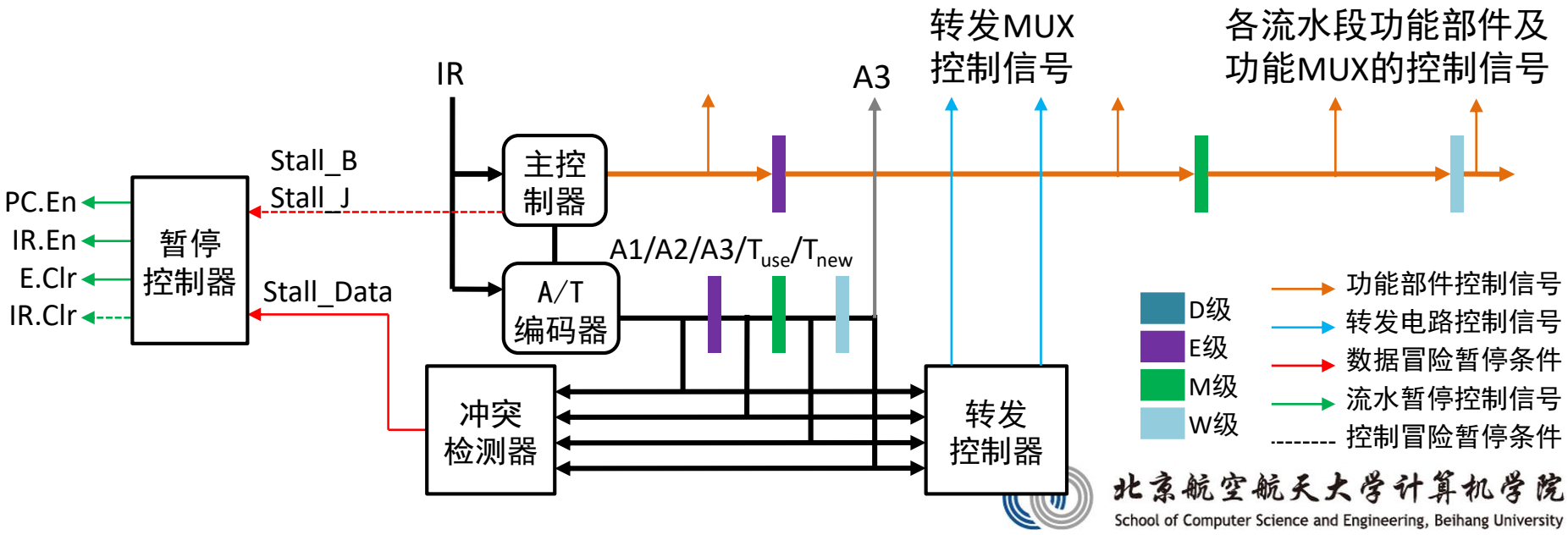
控制层1/2

- 控制层：面向指令的控制流，是“策略与机制分离”的策略部分
  - 主控制器：译码产生指令变量、功能部件控制信号、功能MUX控制信号、B类/J类指令的暂停条件
  - A/T编码器：
    - 根据指令变量，将rs、rt、rd、+31等，转换为A1、A2和A3
    - 根据指令变量，产生T_{use}变量和T_{new}编码值



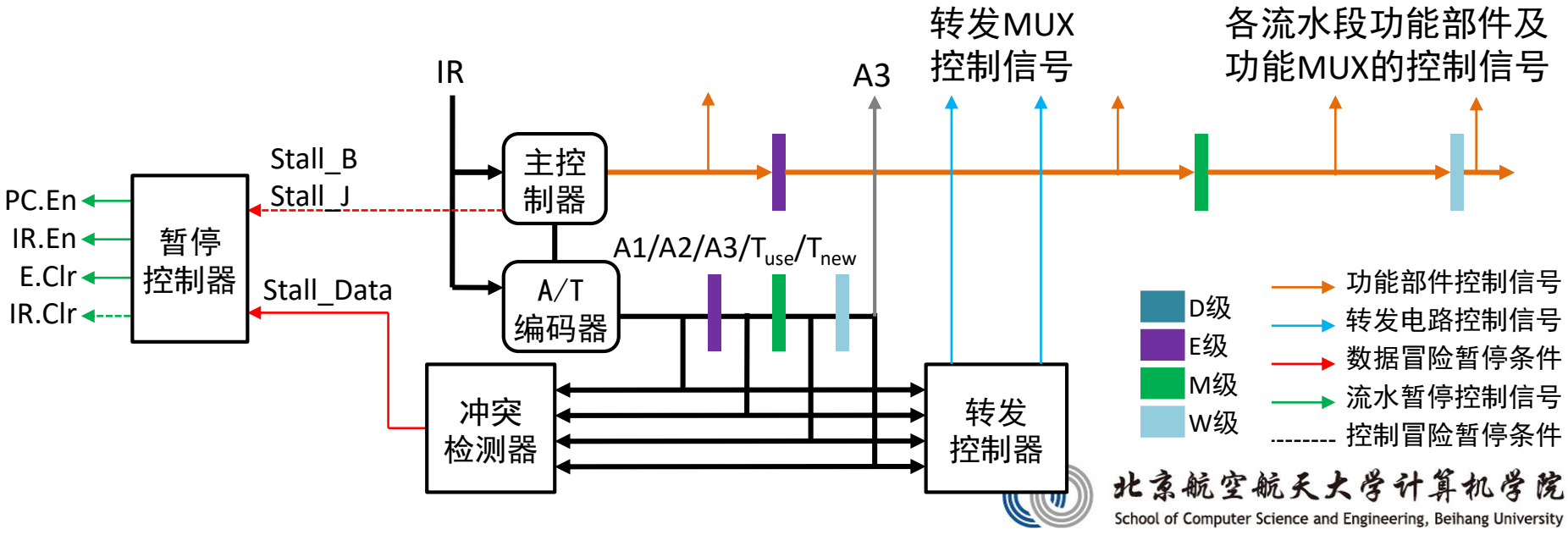
控制层2/2

- 控制层：面向指令的控制流，是“策略与机制分离”的策略部分
  - 冲突检测器：将D级指令信息分别与E、M、W级指令信息进行比较，产生与数据相关的暂停条件
  - 转发控制器：控制各转发MUX，将最新数据转发至各个需求者
  - 暂停控制器：根据主控制器和冲突检测器的暂停条件，产生流水寄存器的控制信号



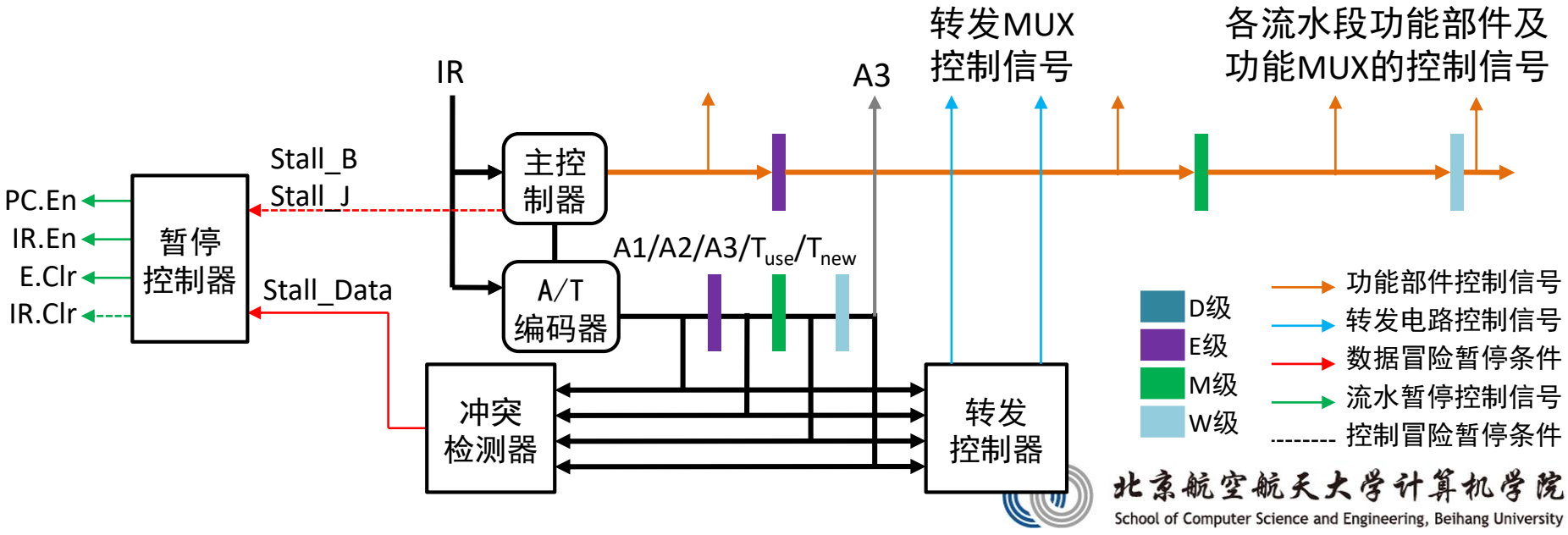
关于5控制器架构

- 思考的角度：如果将控制本身视为系统，则可发现该系统能够进一步分解为多种不同类型的控制信息以及中间状态。
- 控制器的设计原则：仍然是“高内聚、低耦合”
  - 该原则在这里具体体现为：通过合理细分，使得各控制器模块具有较强的功能聚合特性，从而更利于理解、设计与工程实施。



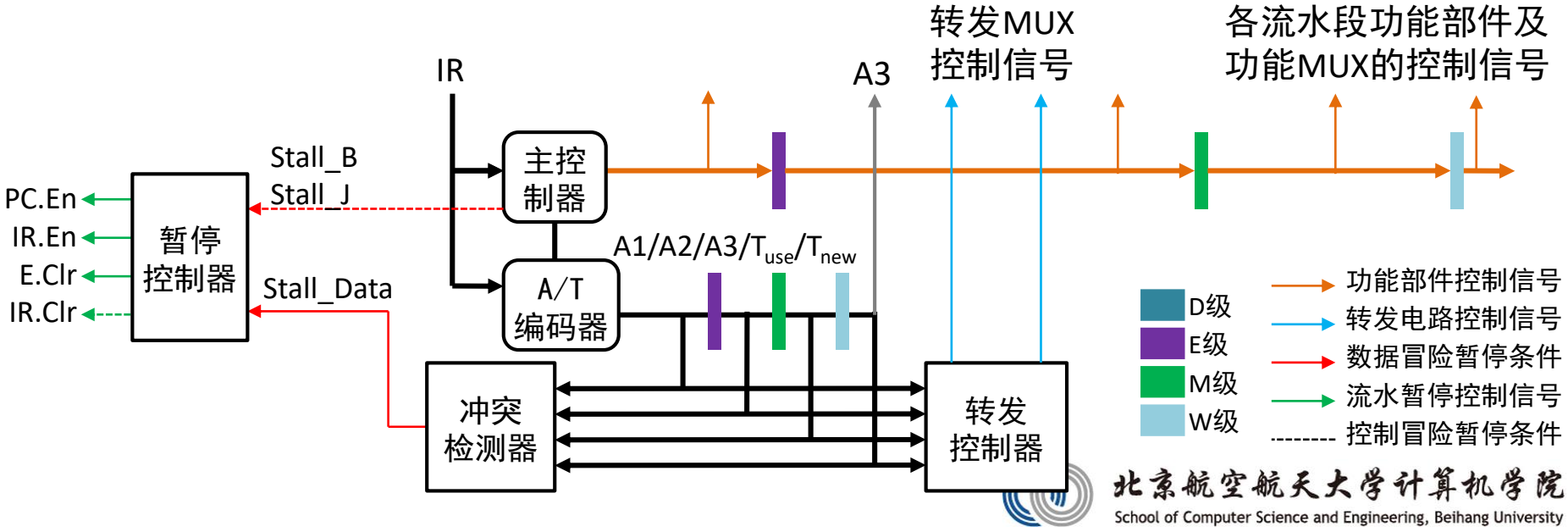
关于5控制器架构

- 冲突检测器、转发控制器
  - 冲突检测与转发控制，虽然都是与数据冒险有关的，但定位不同
    - 冲突检测：发现必须暂停的条件
    - 转发控制：选择正确的转发源
  - 由于定位不同，两者之间并无需要传递的信息
  - 两者的内部逻辑均较为庞大
  - 综合以上，将两者分离实现更为合理



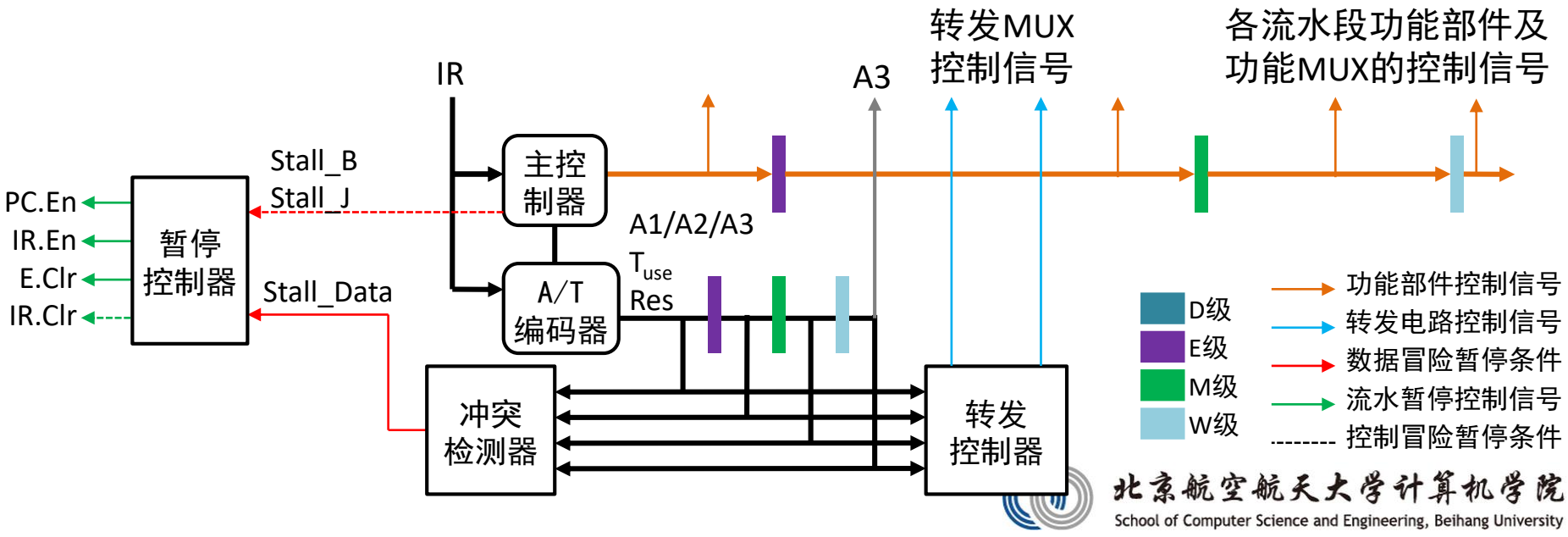
关于5控制器架构

- 暂停控制器
  - 虽然其内部仅包含几行与寄存器使能和清除有关的表达式，但它的输入条件分别是b、j类指令造成的（前提是如果不支持延迟槽指令），以及数据相关造成的
  - 这两种条件源自不同控制器，故将暂停控制独立实现较为合理
  - 此外，假设今后还会有来自第3个源的暂停条件，或者暂停操作本身非常复杂，则分离设计更利于今后的可能扩展
    - 冲突检测与暂停控制的关系非常类似于医生诊断与药房取药



关于5控制器架构

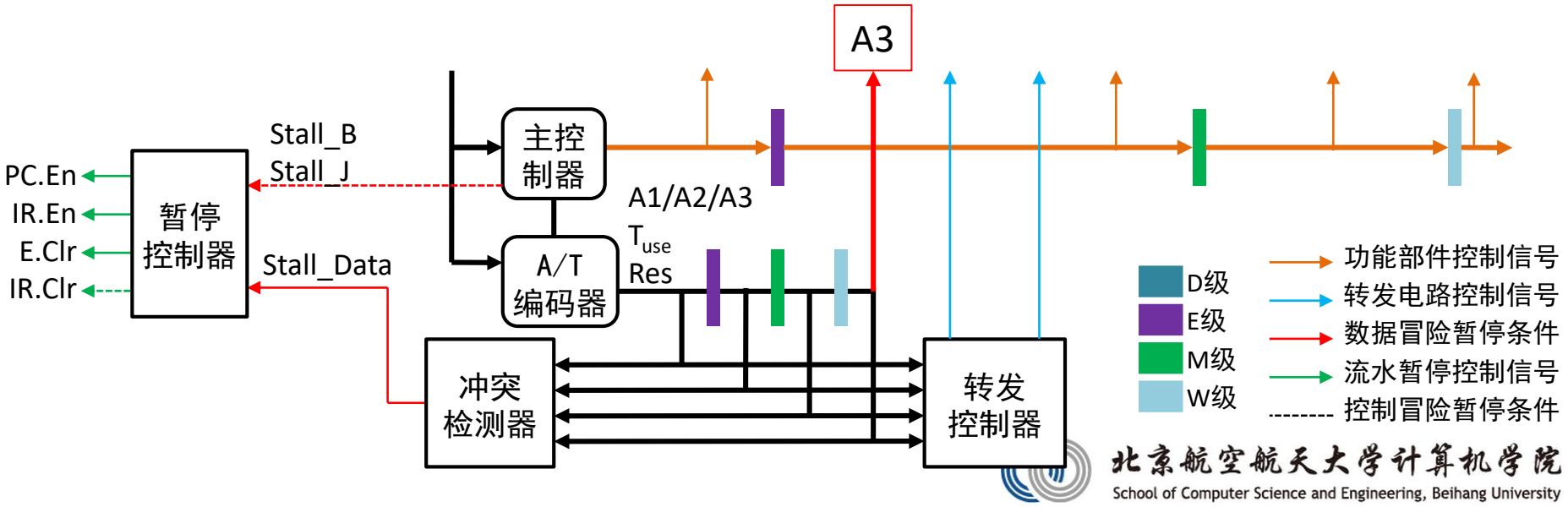
- A/T编码器
  - A编码：无论采用教科书的方案还是PPT的方案，都需对rt、rd域进行二次识别，这就是PPT方案干脆将其变换为A3以消除二义性的原因
    - A1和A2不需要逻辑变换，仅仅是把rs和rt进行命名变换即可
  - T编码：为支持数据相关分析与处理，必须有T_{new}和T_{use}
    - 实际使用时，用Res来替代T_{new}
  - 考虑到两类编码均与指令变量密切相关，因此两类编码合并在一个模块



关于A1、A2和A3

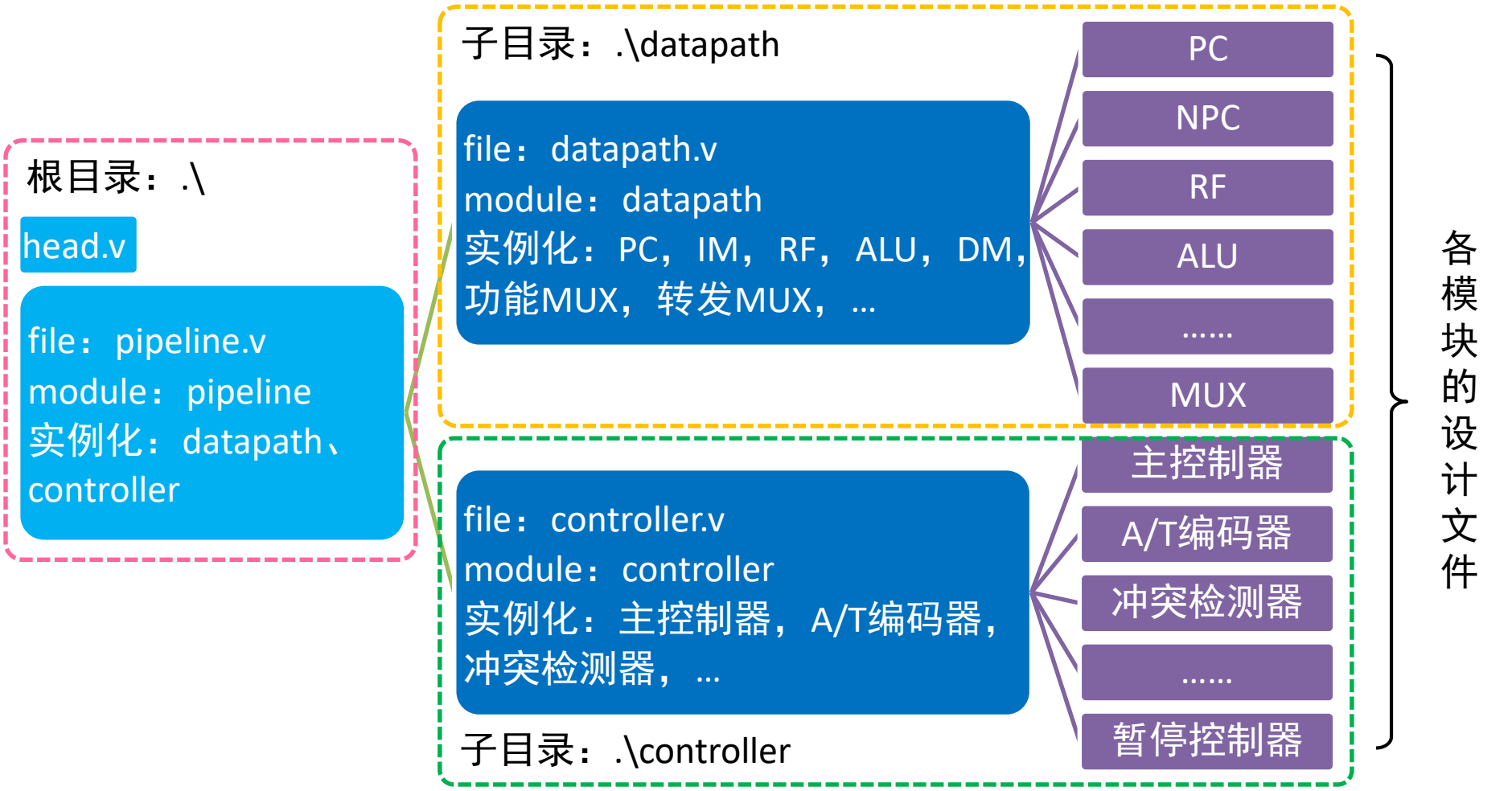
在L15中，A1、A2和A3均位于数据通路中

- 本PPT将三者部署在控制层，理由如下：
  - 1）在原方案的数据通路中，A1、A2、A3仅仅与RF的A1端、A2端和A3端连接，而对于其他功能部件没有任何价值。
    - 除此，三者在数据通路建模中再无其他价值，反而导致设计表变大
  - 2）事实上，这3个信息主要用于数据相关分析。为此，原方案必须在数据层与控制层之间设置大量的接口信号
  - 3）综上，将三者部署在控制层，则仅需要保留A3这一唯一接口即可，使得数据层与控制层耦合更少，更符合“高内聚、低耦合”的设计原则



代码架构

- 建议采用层次化目录
  - 为方便引用，将包含有各种宏定义的head.v部署在工程的根目录



模块实例化注意事项

- 模块间的信号应显式定义后再使用

```
module pipeline( clk, rst ) ;

   wire [31:0]  instr ;
   wire [4:0]   A3 ;
   wire         RFWr, DMWr ;
   wire         PCEn, DEn, Eclr ;
   ……

   datapath U_datapath( …, instr, A3, RFWr, … ) ;
   controller U_controller( …, instr, A3, RFWr, … ) ;

endmodule
```

- 注意信号名在各模块中的正确顺序

MUX设计

- 可以在一个设计文件中定义多种MUX

```
// mulTIPSlexor.v

// 2-1
module MUX2to1( in0, in1, sel, out ) ;
…
endmodule

// 3-1
module MUX3to1( in0, in1, in2, sel, out ) ;
…
endmodule

// 4-1
module MUX4to1( in0, in1, in2, in3, sel, out ) ;
…
endmodule
```

MUX设计

- 为了灵活支持多种位宽的MUX，应使用parameter定义位宽

```
module MUX3to1( in0, in1, in2, sel, out ) ;
    parameter WIDTH_DATA = 32 ;
    input   [WIDTH_DATA:1]          in0;
    …
    output  [WIDTH_DATA:1]          out;

    assign out = (sel==2'b10) ? in2 :
                 (sel==2'b01) ? in1 :
                                in0 ;
endmodule
```

- 实例化时，用defparam设置实际参数

```
MUX3to1 MRFA3(ir[15:11], ir[20:16], 5'd31, A3Sel, A3) ;
    defparam    MRFA3.WIDTH_DATA = 5 ;

MUX3to1 MRFWD(AO_W, DR_W, PC4_W, WDSel, WD) ;
    defparam    MRFWD.WIDTH_DATA = 32 ;
```

MUX设计

- 也可以用always语句建模MUX，但要注意：
  - 1）always敏感表必须对所有信号敏感。always(in0, in1, in2…, out)也可以
  - 2）always中不能有未赋值的空分支

```
module MUX3to1( in0, in1, in2, sel, out ) ;
    reg [31:0] out ;
    always @(*)
        case(sel)
                2: out <= in2;
                1: out <= in1;
                default: out <= in0;
        endcase
endmodule
```

正确

```
always @(*)
    case(sel)
        2: out <= in2;
        1: out <= in1;
        0: out <= in0;
    endcase
```

错误

MUX设计

- 代码的错误在于：存在sel[1:0]为2'b11的无赋值分支，于是综合器会认为在那种情况下变量保持不变。这就相当于被补全了一句：

```
always @(*)
    case(sel)
        2: out <= in2;
        1: out <= in1;
        0: out <= in0;
    endcase
```

错误

```
always @(*)
    case(sel)
        2: out <= in2;
        1: out <= in1;
        0: out <= in0;
       default : out <= out ;
    endcase
```

- 这将带来一个隐患：原本希望设计的是纯组合逻辑，但最终综合器却产生了“组合逻辑+锁存器(latch)”的结果
  - 如果sel为2'b11，则out会输出上次的结果。由于MUX的外部控制逻辑不应产生2'b11，并且一旦产生了则首先说明控制逻辑出错了。故不会仅由该隐患就导致运行错误。
  - 但类似写法也许在其他设计中就会导致错误了。

TIPS 慎用always建模组合逻辑。最好只用always建模时序逻辑。

流水线寄存器实例化

- 流水线寄存器有多种实现方式，其中比较直观的是采用实例化方式。

```
module datapath( …… ) ;
  …
  wire [31:0] …, AO_M, AO_W, DR_W ;

  // E级功能部件
  ALU      U_ALU(…, AO, …) ;

  // M级流水寄存器
  R_Pipe   PIPE_AO_M(AO, AO_M, clk) ;
    defparam PIPE_AO_M.WIDTH_DATA = 32 ;

  // M级功能部件
  DM       U_DM(…, AO_M, DMO, … ) ;

  // W级流水寄存器
  R_Pipe   PIPE_AO_W(DMO, DR_W, clk) ;
    defparam PIPE_AO_W.WIDTH_DATA = 32 ;
  …
```

```
module R_Pipe( di, do, clk ) ;
  parameter WIDTH_DATA = 32 ;
  …
  always @(…)
    do <= di ;
endmodule
```

TIPS

实例化方式实现流水线寄存器，可以使得代码结构布局尽可能与设计布局保持一致。